PMR: This project was easy with the help of the module since it led me through the steps of how to modify the program. I enjoyed this easy assignment and it was a break from writing extensive code. I look forward to using this new skill.

**ANNUALFUELUSEV2**

/\*\*

\* Write a description of class AnnualFuelUse here.

\*

\* @author Anika Jallipalli

\* @version 2/16/2020

\*/

import java.io.\*;

import java.text.\*;

import java.util.\*;

public class AnnualFuelUseV2

{

//instance variables

private int fillup, day, startmiles, endmiles;

private double dis, gallonsused, mpg, prices, cost;

//constructor for objects

AnnualFuelUseV2(int fillup, int day, int startmiles, int endmiles, double gallonsused, double prices)

{

this.fillup = fillup;

this.day = day;

this.startmiles = startmiles;

this.endmiles = endmiles;

this.gallonsused = gallonsused;

this.prices = prices;

dis = 0.0;

mpg = 0.0;

cost = 0.0;

}

//calculate distance

public void calcDistance()

{

dis = this.endmiles - this.startmiles;

}

//return distance

public double getDistance()

{

return dis;

}

//calculate mpg

public void calcMPG()

{

mpg = dis / this.gallonsused;

}

//return mpg

public double getMPG()

{

return mpg;

}

//calculate cost

public void calcCost()

{

cost = this.prices \* this.gallonsused;

}

//return cost

public double getCost()

{

return cost;

}

//return all the other values

public int getFill()

{

return this.fillup;

}

public int getDay()

{

return this.day;

}

public int getStart()

{

return this.startmiles;

}

public int getEnd()

{

return this.endmiles;

}

public double getGallons()

{

return this.gallonsused;

}

public double getPrice()

{

return this.prices;

}

}

**CATAPULTTESTERV2**

/\*\*

\* This class tests the Catapult class.

\*

\* @author Anika Jallipalli

\* @version 2/16/2020

\*/

import java.util.\*;

import java.text.\*;

public class CatapultTesterV2

{

public static void main(String[] args)

{

int [] speed = {20,25,30,35,40,45,50};

int [] angle = {25,30,35,40,45,50};

CatapultV2 catapult = new CatapultV2(speed, angle);

catapult.calcDistance();

System.out.println(" Projectile Distance (feet) ");

System.out.println(" MPH 25 deg 30 deg 35 deg 40 deg 45 deg 50 deg");

System.out.println("=================================================================================================================");

catapult.print();

}

}

**ANNUALFUELUSETESTERV2**

/\*\*

\* Write a description of class AnnualFuelUseTester here.

\*

\* @author Anika Jallipalli

\* @version 2/16/2020

\*/

import java.io.\*;

import java.text.\*;

import java.util.\*;

public class AnnualFuelUseTesterV2

{

public static void main(String[] args)

{

System.out.printf("%s \n","Fill Up Days Start Miles End Miles Distance Gallons Used MPG Price Cost ");

double disMax, disMin, disSum, mpgMax, mpgMin, mpgSum, priceMax, priceMin, priceSum, absMax, absMin, costSum, daySum, gallonsSum;

disSum = 0.0;

disMax = 0.0;

disMin = 0.0;

mpgMax = 0.0;

mpgMin = 0.0;

mpgSum = 0.0;

priceMax = 0.0;

priceMin = 0.0;

priceSum = 0.0;

costSum = 0.0;

daySum = 0.0;

gallonsSum = 0.0;

AnnualFuelUseV2[] fillups = {

new AnnualFuelUseV2(1, 1, 45023, 45231, 10.00, 2.95),

new AnnualFuelUseV2(2, 4, 45231, 45480, 11.70, 2.99),

new AnnualFuelUseV2(3, 8, 45480, 45659, 9.30, 3.03),

new AnnualFuelUseV2(4, 13, 45659, 45961, 14.90, 3.01)

};

//call methods

for(int i = 0; i < fillups.length; i++)

{

fillups[i].calcDistance();

fillups[i].calcMPG();

fillups[i].calcCost();

}

//print the results

for(int i = 0; i < fillups.length; i++)

{

System.out.printf("%3s %9s %11s %12s %10.0f %13.2f %8.1f %7.2f %6.2f \n",

fillups[i].getFill(), fillups[i].getDay(),

fillups[i].getStart(), fillups[i].getEnd(),

fillups[i].getDistance(), fillups[i].getGallons(),

fillups[i].getMPG(), fillups[i].getPrice(),

fillups[i].getCost());

}

//calculate totals for printing and calculating min and max

for(int i = 0; i < fillups.length; i++)

{

disSum += fillups[i].getDistance();

mpgSum += fillups[i].getMPG();

priceSum += fillups[i].getPrice();

costSum += fillups[i].getCost();

daySum += fillups[i].getDay();

gallonsSum += fillups[i].getGallons();

}

//calculate min and max

absMax = Double.MIN\_VALUE;

absMin = Double.MAX\_VALUE;

mpgMax = absMax;

mpgMin = absMin;

disMax = absMax;

disMin = absMin;

priceMax = absMax;

priceMin = absMin;

for(int i = 0; i < fillups.length; i++)

{

if((fillups[i].getDistance()) > disMax)

{

disMax = fillups[i].getDistance();

}

if((fillups[i].getDistance()) < disMin)

{

disMin = fillups[i].getDistance();

}

if((fillups[i].getMPG()) > mpgMax)

{

mpgMax = fillups[i].getMPG();

}

if((fillups[i].getMPG()) < mpgMin)

{

mpgMin = fillups[i].getMPG();

}

if((fillups[i].getPrice()) > priceMax)

{

priceMax = fillups[i].getPrice();

}

if((fillups[i].getPrice()) < priceMin)

{

priceMin = fillups[i].getPrice();

}

}

//print the max and min

System.out.println();

System.out.printf("%s %40.0f %22.1f %7.2f \n", "Minimum:", disMin, mpgMin, priceMin);

System.out.printf("%s %40.0f %22.1f %7.2f \n", "Maximum:", disMax, mpgMax, priceMax);

System.out.println();

//print total

System.out.printf("%s %41.0f %13.2f %23.2f \n", "Totals:", disSum, gallonsSum, costSum);

//calculate annual projections

double distanceProjection, gallonsusedProjection, mpgProjection, costProjection;

double a = 31 / daySum;

a \*= 12;

distanceProjection = a \* disSum;

gallonsusedProjection = a \* gallonsSum;

mpgProjection = a \* mpgSum;

costProjection = a \* costSum;

//print annual projection

System.out.printf("%s %30.0f %13.2f %8.2f %14.2f \n", "Annual Projection:", distanceProjection, gallonsusedProjection, mpgProjection, costProjection);

}

}